VŠB - Technická univerzita Ostrava

Fakulta elektrotechniky a informatiky

Katedra informatiky

Absolvování individuální odborné praxe

Individual Professional Practice in the Company

2011

Achiles Savidis

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

V Ostravě dne 4. 5. 2011 ……

Abstrakt

Odborná praxe je skvělým prostředkem pro získání praktických znalostí a zkušeností. Na rozdíl od klasické bakalářské práce se jedná o práci na něčem, co budou zákazníci skutečně používat. Navíc přináší jinak zkušenost práce v týmu, kterou by student jinak získal až po nástupu do práce.

Pracoval jsem jako vývojář v jazyku C#. Větší část odborné praxe jsem strávil na vývoji fakturačního systému pro utilitní společnosti. Byl jsem součástí týmu složeného z patnácti členů včetně analytiků a vývojářů.

Klíčová slova

Odborná praxe, práce v týmu, vývoj software, .NET, C#

Abstract

Individual Professional Practice is great way to achieve practical knowladge and experience. Unlike the typical Bachelor thesis Individual Professional Practice work on something ment to be realy used by the customers. It also brings irreplaceable teamwork experience.

I worked as developer in C# language. Most of the time I spent developing billing system for utility companies. I was member of team composed of fifteen people including analyst and developers.

Keywords

Individual Professional Practice, teamwork, software development, .NET, C#

Seznam použitých zkratek a symbolů

WCF	- Windows Communication Foundation
WPF	- Windows Presentation Foundation
XML	- eXtensible Markup Language
XAML	- eXtensible Application Markup Language
HTML	- Hyper Text Markup Language
ISU	- Informační Systém Utility
SQL	- Structured Query Language

Obsah

1 Úvod

V následujícím textu se seznámíme s tím, že jsem pracoval ve firmě D3Soft s.r.o. a také pár informací o ní. Dozvíme se, že jsem pracoval jako vývojář - programátor v jazyku C# a co bylo mou pracovní náplní. Nepřehlédneme ani dva klient-server projekty, na kterých jsem se také podílel. Jedním je ISU - fakturační systém pro utilitní společnosti a druhým Task Manager – aplikace pro zařazení úkolů do fronty a jejich automatické zpracování.

Protože D3Soft s.r.o. se snaží využívat nové technologie a udržovat krok s dobou, také si některé technologie použité v projektu představíme a projdeme. Například Windows Communication Foundation a Windows Presentation Foundation. Obě jsou vydány společností Microsoft jako součást .NET Frameworku. Architektura aplikace byla na těchto technologiích postavena, a proto si zaslouží své místo v této bakalářské práci.

Nahlédneme krátce do struktury samotné aplikace. Ukážeme si, že se dělí na klientskou část a na serverovou část. Server je navíc rozdělen do tří vrstev, kde každá vrstva plní svou stanovenou úlohu. Díky dodržován pravidel, která tato architektura vytváří pak můžeme jednoduše jednu vrstvu změnit aniž by bylo zapotřebí měnit vrstvu jinou. Taková modularita potom přináší vývojářům mnoho užitku.

Projdeme si dva konkrétní příklady, které jsem v rámci odborné praxe vypracoval – číselník produktu a aktivity se zákazníkem nebo dodavatelem. Oba zmíněné úkoly se týkají projektu ISU a jsou jeho rozšířením. Zároveň to jsou první dva úkoly, které mi byly v rámci odborné praxe zadány.

Na závěr dáme prostor mému osobnímu hodnocení a dojmům. I tomu, zda mě tato praxe od programování odradila nebo spíše vzbudila můj zájem a touhu učit se dalším programátorským dovednostem.

2 D3Soft s.r.o.

D3Soft s.r.o. [1] je středně velká česká softwarová firma s více než desetiletou historií. Zaměřují se na podporu firemních procesů. Jejich produkty jsou vhodné pro malé, střední i velké firmy. Své zákazníky má nejen v české republice, ale také v řadě jiných zemí střední Evropy jako například Německo nebo Polsko.

D3Soft s.r.o. je také partnerem společnosti Microsoft. Nedávná změna podmínek společnosti Microsoft však způsobila, že své členství v rámci Microsoft Partner Network programu pro úroveň Gold Software Development museli vývojáři společnosti D3Soft s.r.o. obhajovat. Tohoto cíle dosáhli úspěšně právě v období mého působení v jejich vývojářském týmu.

Obrázek 1: D3Soft logo

3 Popis pracovního zařazení

3.1 Pozice

Ihned od začátku jsem nastoupil jako externista na pozici vývojář – programátor. Na této pozici jsem zůstal až do konce odborné praxe. Externistou rozumíme, že jsem nepracoval na klasickou zaměstnaneckou smlouvu, ale na dohodu o provedení práce. Praxi jsem vykonával v sídle firmy spolu se všemi dalšími členy týmu. V zimním semestru jsem docházel třikrát týdně, v letním semestru se mi dařilo docházet pětkrát týdně.

Má práce zahrnovala přijetí úkolu od analytika, jeho implementaci a odevzdání k testování. Po otestování se někdy úkol vrátil zpět, a s ním i popis chyb, které odhalilo testování a bylo potřeba je opravit.

3.2 Projekty

V průběhu odborné praxe jsem se podílel na vývoji dvou projektů. Prvním je ISU [2], druhým pak Task Manager. Oba programy jsou klient-server aplikace s tenkým klientem.

ISU je aplikací vcelku rozsáhlou. Pracuje se na něm už více než rok a během mé praxe tým tvořilo 15 členů. To mělo za následek aplikaci skládající se z více než stovky dílčích projektů. Tato aplikace je kvůli své rozsáhlosti stále ještě v procesu vývoje a nasazování.

Task Manager by se rozsahem dal zařadit do kategorie střední. Je složen z přibližně čtyřiceti dílčích projektů. Pracovali jsme na něm v týmu tvořeném více než pěti lidmi, přičemž já osobně jsem spolupracoval s dalšími třemi programátory, jedním analytikem a jedním grafikem. Task Manager se začal vyvíjet přibližně měsíc před mým odchodem a v době psaní této práce je stále vyvíjen.

Dílčími projekty rozumíme projekty Visual Studia, kterých lze v jedné aplikaci mít více než jednu. Projekty mohou mít vytvořenou vzájemnou vazbu nebo nemusí. Tato vazba pak umožňuje používat v jenom projektu objekty jiného projektu. Významem tohoto dělení je izolovanost kódu, kde to je potřeba nebo vhodné a zároveň lepší organizovanost a přehlednost. Pomocí dělení do těchto projektů je pak také dosaženo třívrstvé architektury, která je rozebírána později.

Na rozsahu těmto aplikacím přidává ještě to, že jsou obě napojeny na další dílčí služby, které běží samostatně. Jejich kód se v samotné aplikaci neobjevuje, pouze tyto služby z aplikace voláme. Snahou je, aby část aplikace, kterou lze univerzálně využít v jiné její části nebo v další aplikaci, byla po zvážení převedena do samostatné služby. Takovou službu pak můžeme dále rozšiřovat a upravovat. Tím je zajištěna univerzálnost.

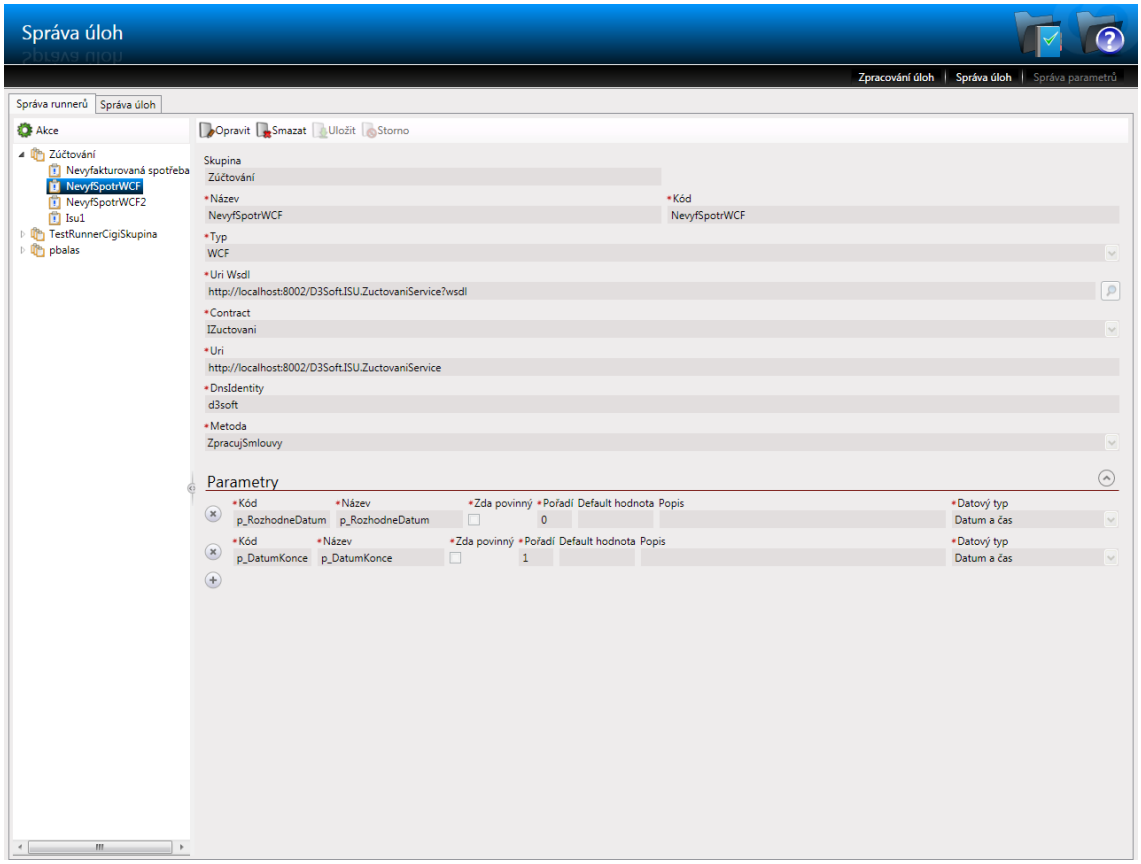
3.2.1 ISU

ISU znamená Informační Systém Utility. Tento produkt je fakturačním a informačním systémem pro utilitní společnosti. Tedy společnosti prodávající elektřinu, plyn nebo jiný zdroj energie.

Při mém nástupu byl tento produkt již ve značně pokročilé fázi vývoje. Díky tomu jsem také mohl některé části kódu převzít z již existující části a upravit je pro aktuální potřebu.

Někdy také docházelo ke změně chování programu nebo k opravě chyb, které odhalilo testování v částech, které byly již dříve naprogramovány někým jiným. V takových případech, pak bylo nutné orientovat se v cizím kódu, což se prokázalo být někdy náročné.

3.2.2 Task Manager



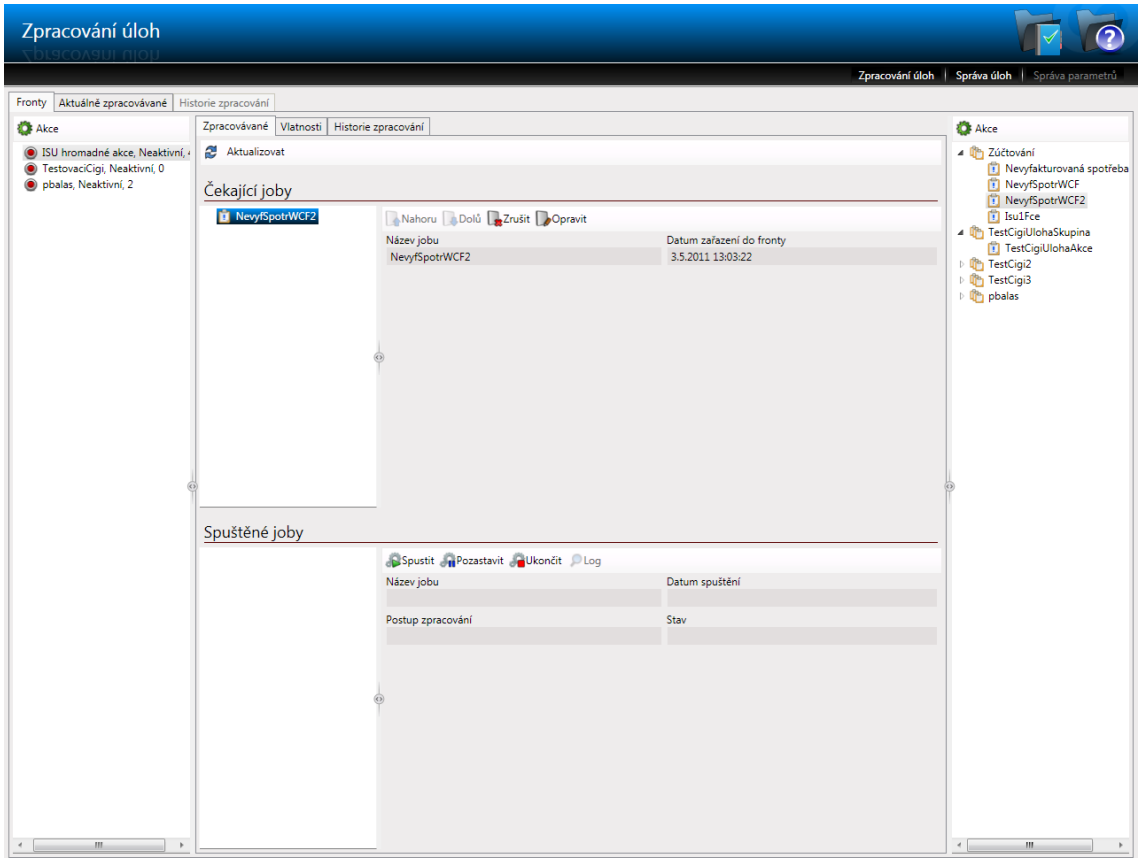
Obrazek 2: Task Manager – správa úloh

Task Manager slouží při zpracovávání úloh, které by uživatel jinak musel zdlouhavě provádět sám ručně, přičemž by do ukončení neměl žádný přehled o aktuálním stavu. Tyto procesy obvykle navíc zahltí spojení k dané službě a tak je třeba čekat až do doby, kdy je úloha dokončena.

V programu Task Manager je možno takové procesy vložit do fronty a ve správný čas ji spustit. Tato fronta běží v novém vláknu, a tudíž nezahltí celou aplikaci, pro kterou jsou úkoly vykonávány, a uživatel může dále pokračovat v práci na jiných úkolech. Průběžně je také vypisován stav úkolů fronty a proto má obsluha dobrý přehled o dané situaci.

U tohoto produktu jsem byl téměř od jeho vzniku. I když jsem se nepodílel přímo na vytváření samotného jádra, byl jsem přítomen u toho, když kolegové řešili některé problémy a tak jsem mohl alespoň málo pochytit i v této oblasti.

Práce na něm nebyla rozdílná od práce, kterou jsem vykonával na projektu ISU, aplikace obsahovala i stejnou strukturu. Přesto však byla práce na nově vznikajícím projektu pro mě velmi zajímavá a pomohla mi vytvořit si lepší přehled ve struktuře aplikace a dalších oblastech.



Obrázek 3: Task Manager – Zpracování úloh

4 Využívané technologie

Následující podkapitoly jsou krátkým přiblížením technologií spadajících do rodiny .NET Framework či způsobem jak nějakou z těchto technologií využívat. Lambda výrazy jsou pak jednou z možností nejen programovacího jazyka C#, který také spadá do .NET Framework a protože je na ně poslední dobou kladen větší důraz, určitě stojí za zmínku.

4.1 Windows Communication Foundation

WCF je aplikační programovací rozhraní v .NET Framework pro vytváření aplikací služeb [3]. Můžeme si tak aplikaci rozdělit na několik služeb, kdy každá služba běží samostatně. Tyto služby ale mohou být různě propojeny a tak navzájem spolupracovat.

Pokud je služba volána, připojujeme se k ní skrze endpoint – koncový bod. Tato služba pak představuje server. Aplikace, která ji volá, pak představuje klienta. Bez ohledu na to zda se jedná o klasickou aplikaci nebo o další službu.

Představme si například, že máme několik aplikací, do kterých je třeba se přihlašovat. Každá aplikace může přihlašování zpracovávat sama o sobě nebo si můžeme vytvořit jednu službu, která bude autentizaci zařizovat. Tato služba není omezená tak, že by mohla být volána jen jednou aplikací, ale každou aplikací, v níž si vytvoříme vazbu na naši službu.

V naší službě si vytvoříme si koncový bod, aby mohla být služba volána. Tento koncový bod je realizován metodou, která má jako vstupní parametry přihlašovací údaje, výstupem je datový typ boolean. Touto cestou služba získá zadané hodnoty, aby s nimi mohla dál pracovat. Dále si vytvoříme napojení do databáze, aby bylo možno získat správná data, a naimplementujeme jejich samotné porovnání. Metoda vrátí true pokud jsou informace shodné, jinak vrátí false.

Naši přihlašovací službu bychom mohli dále rozšiřovat a přidávat další metody, které by mohy být skryté anebo tvořit koncové body. Možnosti druhů služeb i jejich rozsahu jsou prakticky neomezené.

4.2 Windows Presentation Foundation

Windows Presentation Foundation je alternativou k WinForms [4]. Umožňuje vytvářet uživatelské rozhraní na vysoké grafické úrovni pro klasické aplikace pro operační systém Microsoft Windows. Díky tomu, že využívá DirectX nám pak oproti WinForms dává lepší možnosti použití 2D a také 3D grafiky včetně různých animací.

Díky značkovacímu jazyku XAML, který používá, pak dochází k oddělení funkčnosti od vzhledu aplikace. XAML na první pohled připomíná značkovací jazyk pro tvorbu webových prezentací HTML. To je dáno tím, že oba vycházejí z XML. XAML však na rozdíl od HTML podporuje mnohem více možností a jiný způsob vykreslování. Proto si i programátoři se znalostí HTML musí ze začátku zvykat na mírně odlišný způsob použití.

Při psaní XAMLu musíme dbát pravidel, která vycházejí z již uvedeného XML. Každá značka začíná znakem ‚<‘ a končí znakem ‚>‘. Všechny značky jsou párové, a proto musí být ukončeny. Ukončovací značka je stejná jako otevírací, ale hned za první znak ‚<‘ přidáme ještě znak lomítko: ‚/‘. Mezi otevírací a ukončovací značky můžeme vkládat další značky, a tím dosahujeme vnořování. Pokud mezi značkami nemáme žádnou další vnořenou, můžeme ji ukončit zvláštním způsobem, kdy namísto otevírací a ukončovací značky použijeme jen jednu,

která stejně jako otevírací značka začíná znakem ‚<', ale na rozdíl od ní končí znaky ‚/>'. Součástí otevírací značky také mohou a někdy dokonce musí být atributy. Ty zapisujeme ve formátu Atribut="Hodnota". Cokoli je napsáno mezi ‚<!--' a ‚-->' je považováno za poznámku a není zobrazeno. Více už vysvětlí následující ukázka:

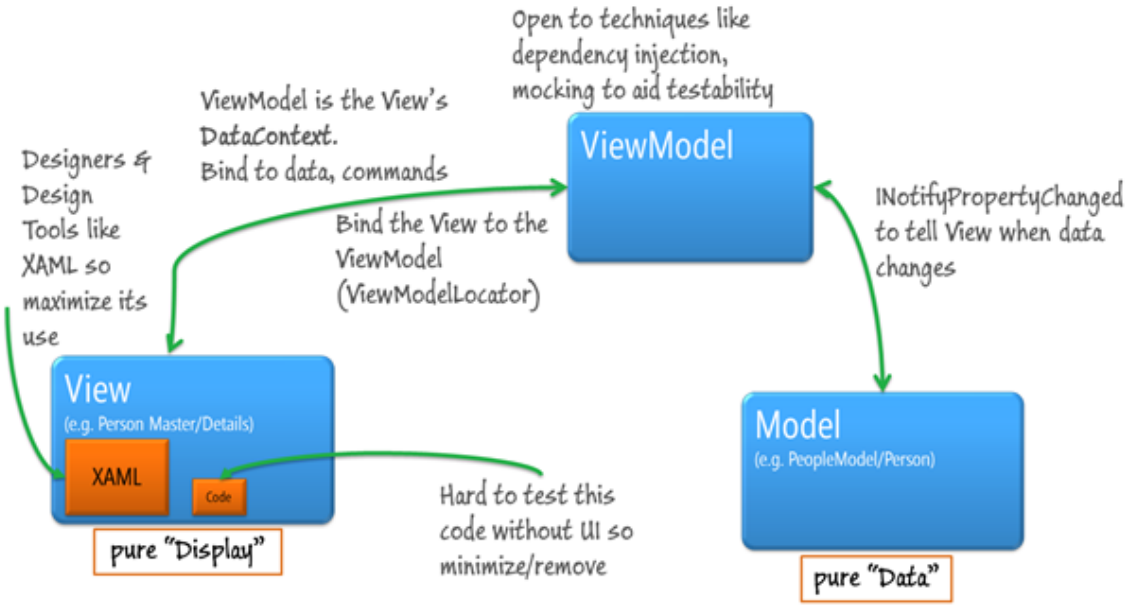
```
<Grid Height="Auto">                                    <!-- otevírací značka -->
   <Label Grid.Row="0" Text="Aktivity" Margin="5" />     <!-- značka, která je rovnou uzavřena-->
</Grid>                                                 <!-- uzavírací značka-->
```

Vývojáři již od začátku počítali s tím, že WPF bude možno používat v kombinaci s různými programovacími jazyky z rodiny .NET. Můžeme se proto setkat s tím, že jedna aplikace využívající WPF je psána v programovacím jazyku C# zatím co jiná, která WPF využívá také je psána v jazyku Visual Basic.

Další výhodou je také to, že společnost Microsoft zahrnula tuto technologii do své platformy Silverlight, která určené převážně pro webové prohlížeče, ale může běžet i v režimu „out of browser". Tím ve velké míře sjednocuje programování pro různé platformy a není tak nutné se pro každé odvětví učit jiný způsob vytváření funkčních aplikací nebo jiný programovací jazyk.

4.2.1 Model-View-ViewModel

Model-View-ViewModel je ve zkratce označován jako MVVM. Je návrhovým vzorem pro WPF aplikace, který vychází ze známějšího návrhového vzoru Model-View-Conroller. Vnáší tak řád do kódu tím, že logicky odděluje různé prvky programovacího jazyka a WPF. Každá část názvu tj. View, Model a ViewModel označuje samostatnou třídu, ve které jsou následně schraňovány objekty, které do dané kategorie spadají. Například odděluje objekty, které jsou propojeny s XAML pomocí bindingu a zahrnuje je do třídy ViewModel. Zatím co metody pro události zůstávaní v kódu na pozadí daného formuláře, který je ve WPF označován jako UserControl.



Obrázek 4: Diagram Model-View-ViewModel

4.3 Lambda výrazy

Lambda výraz bývá označován také jako lambda kalkul. V angličtině bývá také někdy psán jako λ-calculul. Lambda výraz představil Alonzo Church v třicátých letech dvacátého století. Součástí jazyka C# se stal ale až ve verzi 3.0, která byla vydána v .NET Frameworku ve verzi 3.5.

Je to formální systém pro definování funkcí, jejich aplikaci a rekurzi. Je pomocí něj možno vytvořit anonymní metody, které obsahují jeden nebo více výrazů. Operátor, který pro lambda výrazy používáme v jazyku C# je =>. Zápis takového výrazu je potom ve tvaru:

```
(vstupní atributy) => výraz
```

V průběhu praxe jsem pro něj našel využití převážně při vyhledávání v listu objektů. Dejme tomu, že máme sadu čísel typu int. Tato čísla vložíme do seznamu intů. Seznam vytvoříme klíčovým slovem list, který je následován lomenými závorkami, uvnitř nichž je uveden datový typ, jehož seznam vytváříme. Vytvoření nové instance takového seznamu čísel bude vypadat následovně:

```
List<int> seznam = new List<int> { 1, 2, 3, 4, 5 };
```

V tomto případu jsme seznam vytvořili z čísel jedna až pět. Datový typ list podporuje metoda jako find, remove a další, které vyžadují jako vstup predikát neboli pravidlo podle kterého má být prvek nalezen. Právě při použití těchto metod je vhodné použít lambda výraz. Pokud tedy chceme najít v seznamu všechna čísla větší než 3 a uložit je do jiného seznamu čísel, zapíšeme to podle následujícího vzoru:

```
seznam.FindAll((vstupní atribut) => (vstupní atribut) operátor (naše hodnota));
```

Pokud aplikujeme uvedený vzor na náš příklad bude zápis odpovídat následujícímu řádku:

```
List<int> vysledek = seznam.FindAll(i => i > 3);
```

Pokud bychom chtěli tuto metodu použít v jazyku C# ve verzi nižší než 3.0 mohli bychom použít delegát. Pro srovnání s lambda výrazem můžeme použít následující výraz, který je zapsán pomocí delegátu:

```
List<int> vysledek = seznam.FindAll(delegate(int i)
{
    return i > 3;
});
```

Z tohoto srovnání vidíme, že lambda výrazy nejen zkracují, ale často také zjednodušují zápis.

5 Struktura aplikace

5.1 Struktura serveru

Oba programy, na kterých jsem měl možnost se podílet, jsou aplikací typu klient-server. Základ je postaven na databázi, kterou také vytvořila společnost Microsoft. Tou je databáze MSSQL. Nad databází pracuje samotný server, který je postaven na třívrstvé architektuře. Třívrstvá architektura, se obvykle skládá z vrstev, které postupně od nejnižší označují názvy: Datová, Business a Prezentační.

Každá vrstva má svůj specifický význam a účel. Cílem tohoto rozdělení je zefektivnit vývoj tím, že zpřehledníme a zjednodušíme samotný kód. Proto je nutné do každé vrstvy umístit vždy takový kód, který do ní patří.

Hlavní význam této struktury ale spočívá v tom, že každá vrstva je nezávislá na ostatních a je možno ji implementovat a testovat samostatně. Pokud tedy jednu změníme podle daných pravidel, celá aplikace bude nadále funkční bez potřeby změnit jakoukoli jinou vrstvu. Dále si popíšeme jednotlivé vrstvy.

5.1.1 Datová vrstva

Datová vrstva bývá také označována jako Data access – vrstva pro přístup k databázi. Zde implementujeme rozhraní pro přístup k uloženým datům a tato implementace se liší podle toho, s jakou konkrétní databází pracujeme, jelikož každá společnost vyvíjející databáze poskytuje SQL jazyk, který se od ostatních SQL pocházející od jiných výrobců liší.

To co však zůstává stejné, je to, jak se získaná data prezentují do vyšších vrstev. Kýžený výsledek se pak dostaví ve chvíli, kdy se rozhodneme přejít na databázi jiného výrobce

5.1.2 Business vrstva

Business vrstva je v angličtině označována také jako business logic. Název je opět příhodný, protože zde provádíme veškerou logiku a operace s daty. V kódu této vrstvy nalezneme jakékoli operace, které se provádějí s načtenými daty před tím, než jsou poslány v již zpracované formě pro zobrazení uživateli.

5.1.3 Prezentační vrstva

Tato vrstva je obvykle uživatelským rozhraním, zahrnuje samotné GUI aplikace. V projektech, na kterých jsem pracoval, však byla tato vrstva využívána jiným způsobem.

Jelikož námi vytvářené aplikace jsou typu klient-server, není možné, aby tato vrstva byla přímo uživatelským rozhraním. Místo toho jsou výstupním bodem WCF implementace služeb. Zde používáme koncové body, o kterých jsem se zmiňoval již dříve v kapitole o WCF.

5.2 Klient

Klientská část aplikací, při jejichž tvorbě jsem se podílel, je označována jako tenký klient. Tím rozumíme to, že klient neprovádí žádné výpočty nebo zpracovávání dat ani nepřistupuje přímo k databázi. Namísto toho pouze data převedeme do takové podoby, aby bylo možné je odeslat na server, kde už probíhá veškeré další zpracovávání.

ISU i Task Manager byly softwarové produkty s přehledným uživatelským rozhraním. Grafika zpříjemňující práci s nimi nebyla nijak rušivá. Ani se zde neobjevovaly žádné zbytečné animace, které by snad v uživatelích vytvářeli dojmy, že si vývojáři chtěli vyzkoušet, co vše s danou technologií vůbec vytvořit lze, ale zapomněli na použitelnost.

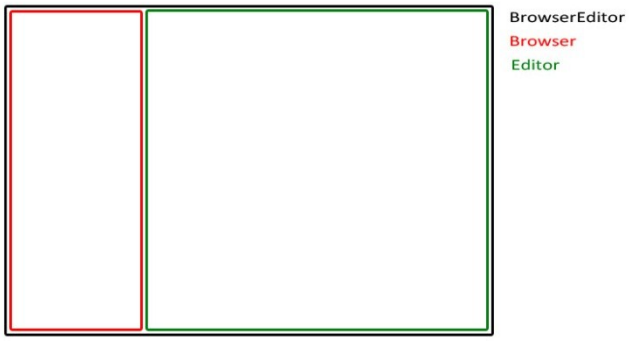
Za zmínku ale stojí prvek, který z plochy aplikace zabíral největší vizuální prostor. Označovali jsme ho jako BrowserEditor.

5.2.1 BrowserEditor

BrowserEditor je základním ovládacím prvkem aplikace, který v sobě spojuje další dva důležité ovládací prvky. Těmi jsou Browser a Editor. Způsob jejich rozložení je znázorněn na obrázku 1.

Browser slouží jako seznam položek právě zobrazovaného typu - například zákazníků. Většinou také obsahoval filtr pro upřesnění daného seznamu, ve kterém by jinak mohlo být obtížné se orientovat.

Editor je uživatelský prvek, který v sobě obsahuje formulář, který nám umožňuje zobrazovat a upravovat konkrétní data, která se vážou k danému prvku v Browseru. Také v sobě obsahuje toolbar, s tlačítky jako „Opravit“, „Storno“, „Uložit“ nebo další tlačítka s předem naprogramovanou akcí.



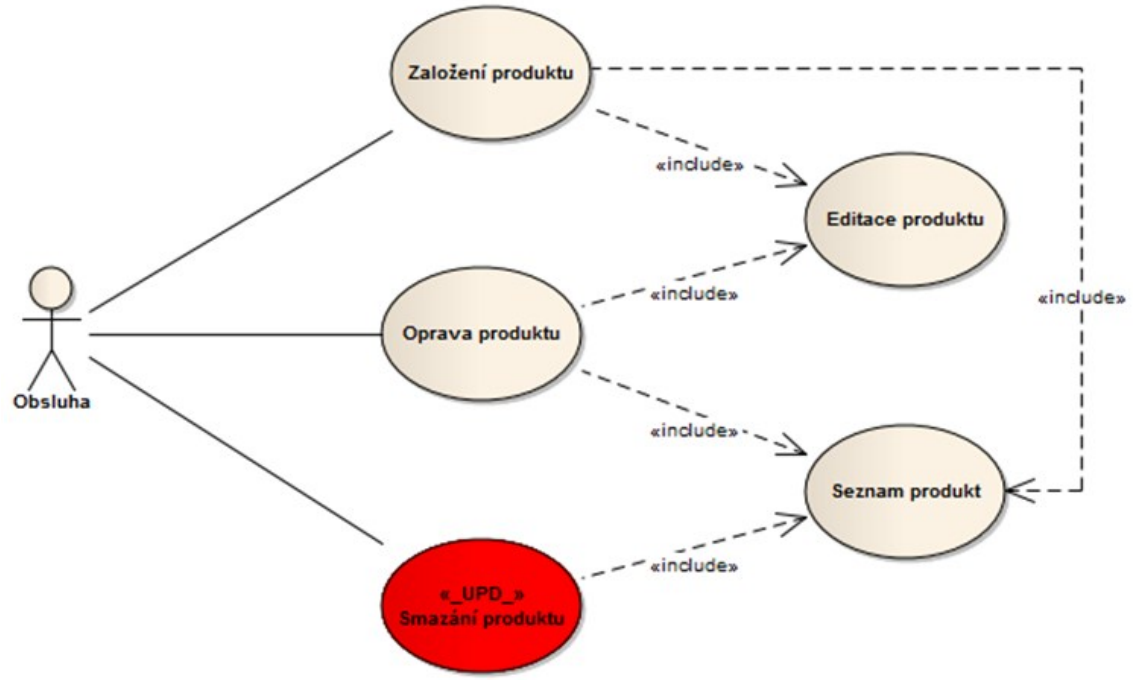
Obrázek 5: Rozložení BrowserEditoru

6 Konkrétní zadané úlohy

6.1 Produkt

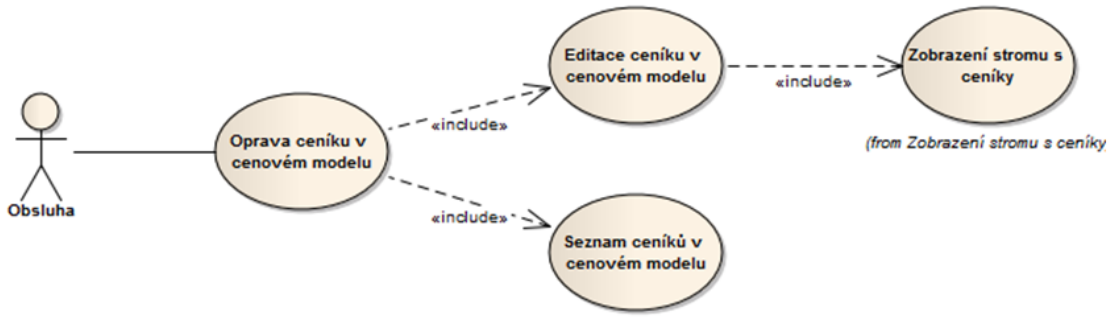
Úvodním úkolem mé odborné praxe a zároveň úvodem do světa WCF a WPF byl úkol „Číselník produktu“.

Na úvod tohoto úkolu jsem dostal vypracovanou analýzu i s jejím kvalitním popisem, která využívala UML diagramů. Tento diagram není výrazně složitý, ale o to lépe popisuje základní vztahy v aplikacích, které jsme vyvíjeli. Tento diagram zahrnuje vztahy pouze v rámci vyvíjeného modulu, ne však vztahy s dalšími částmi aplikace, které pro řešení tohoto problému nejsou zapotřebí, protože se tato sekce jeví jako samostatný celek.

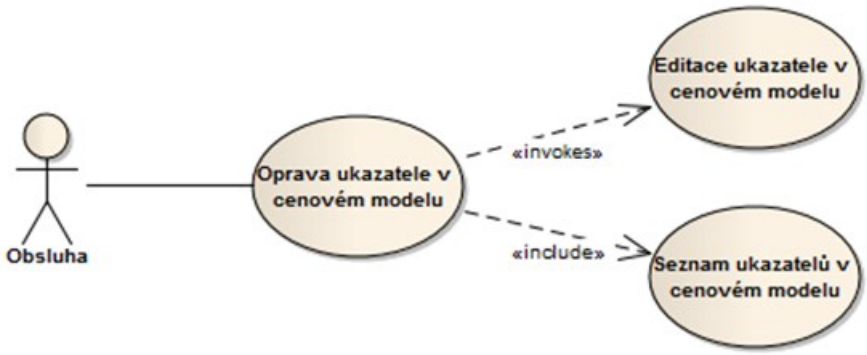


Obrázek 6: Základní UML diagram pro produkt

Protože řešení bylo třeba rozdělit do tří základních celků, následovaly ještě dva doplňující diagramy pro zbývající dvě sekce, které byly označeny jako „ceníky a číselníky“ a „ukazatele“:



Obrázek 7: UML diagram – produkt – cenový model



Obrázek 8: UML diagram – produkt – ukazatele

Řešením tohoto úkolu bylo rozhraní pro prohlížení a úpravu produktu, který však mohl mít různé vlastnosti a tudíž i různé styly zobrazení v závislosti na tom jaké hodnoty obsluha zadala. To znamenalo, že pokud byl ceník správného typu, cenový model a ukazatele se zobrazovaly. Pokud však byl jiného typu, tyto informace produkt neobsahoval, a proto nebylo možné je zobrazit.

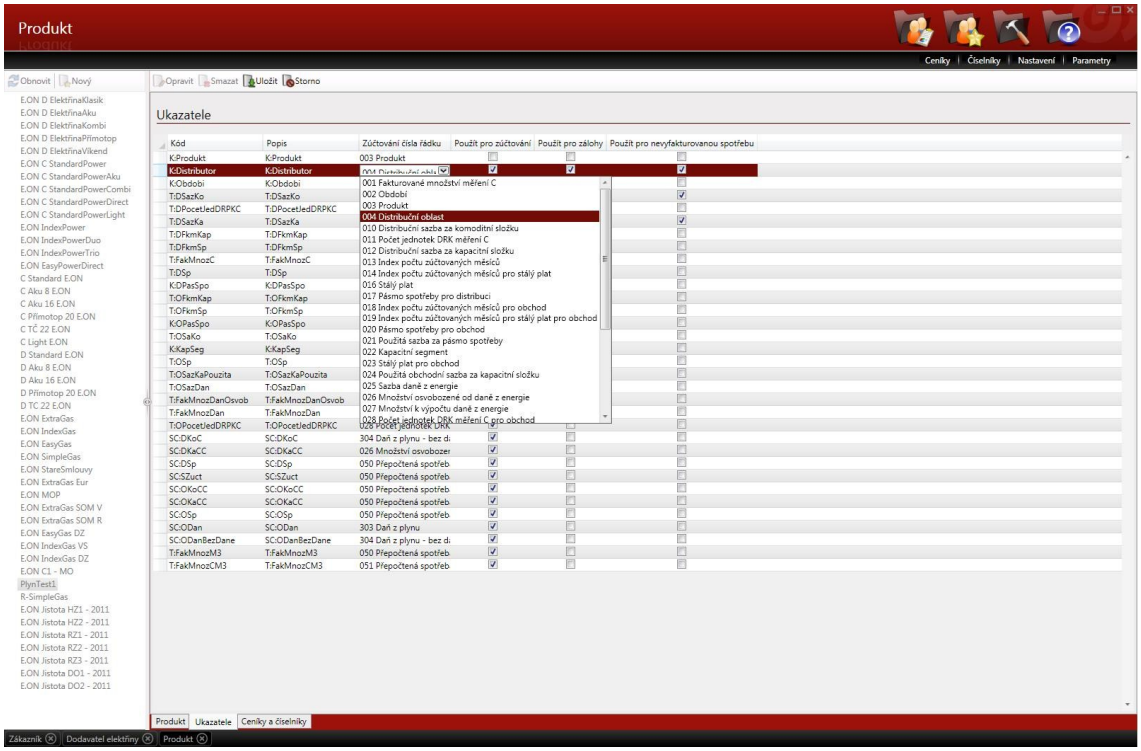
Pro zobrazování jsem využil komponentu TabControl, která umožňuje vytvářet sadu stránek se záložkami. Základní informace byly vždy zobrazeny na první záložce. Pokud byla další data zobrazena, zobrazily se uživateli další dvě záložky s dodatečnými daty. Načíst všechna data by bylo časově náročné, a proto jsem přistoupil k řešení, kdy načítám pouze aktuální vybranou záložku.

Poprvé jsem se také seznámil s ovládacím prvkem WPF, který se jmenuje DataGrid a slouží k vypisování tabulkových hodnot. Tentokrát však nešlo jen o vypisování, ale také o jejich editaci. Editaci takovou, že bylo třeba do tohoto DataGridu vložit další formulářové prvky jako ComboBox či CheckBox jak je to ukázáno na obrázku.

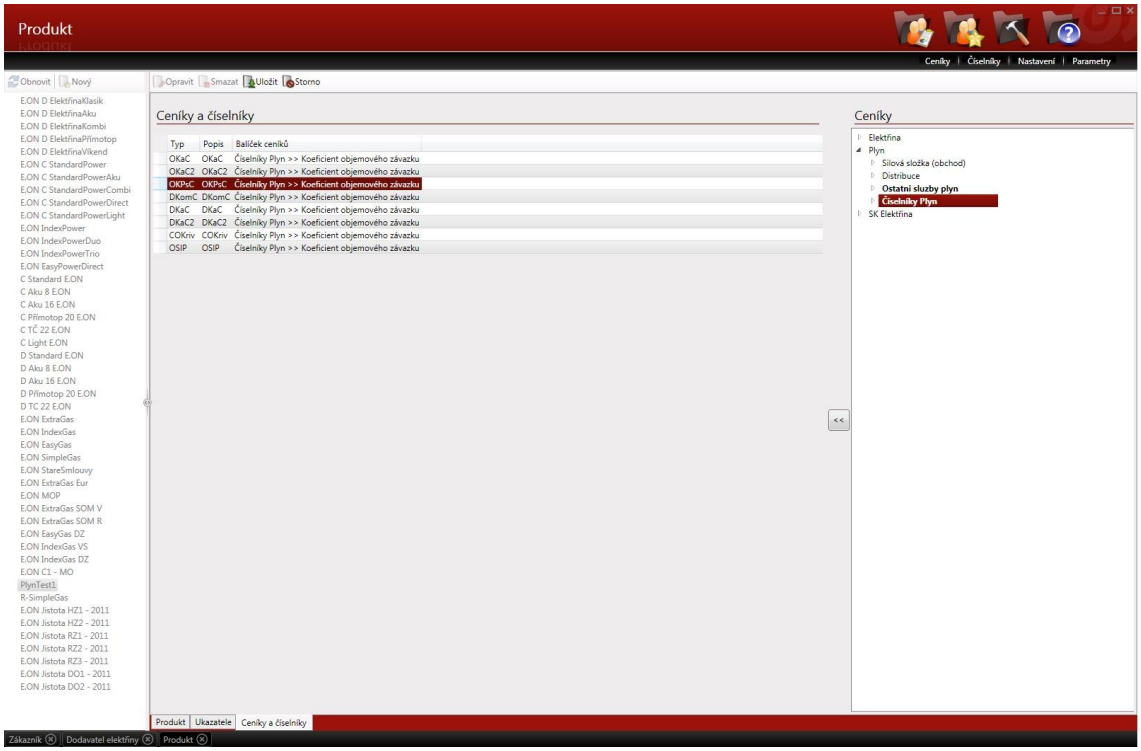
Také nastal případ vhodný pro použití jednoduchého lambda výrazu, jehož účelem bylo vymazat položky listu, které jsme nechtěli zobrazit, nebo najít nějaké podle pravidla

stanoveného lambda výrazem. Toto použití odpovídá příkladu, který jsem uvedl již v kapitole o lambda výrazech.

Tento příklad použití není nijak náročný. I když jsem však využil i výraz komplikovanějšího, je konkrétně tento vhodný jako ukázkový příklad právě proto, že takhle byl nejčastěji využíván ve vyvíjených aplikacích.



Obrázek 9: Náhled produktu – karta ukazatele



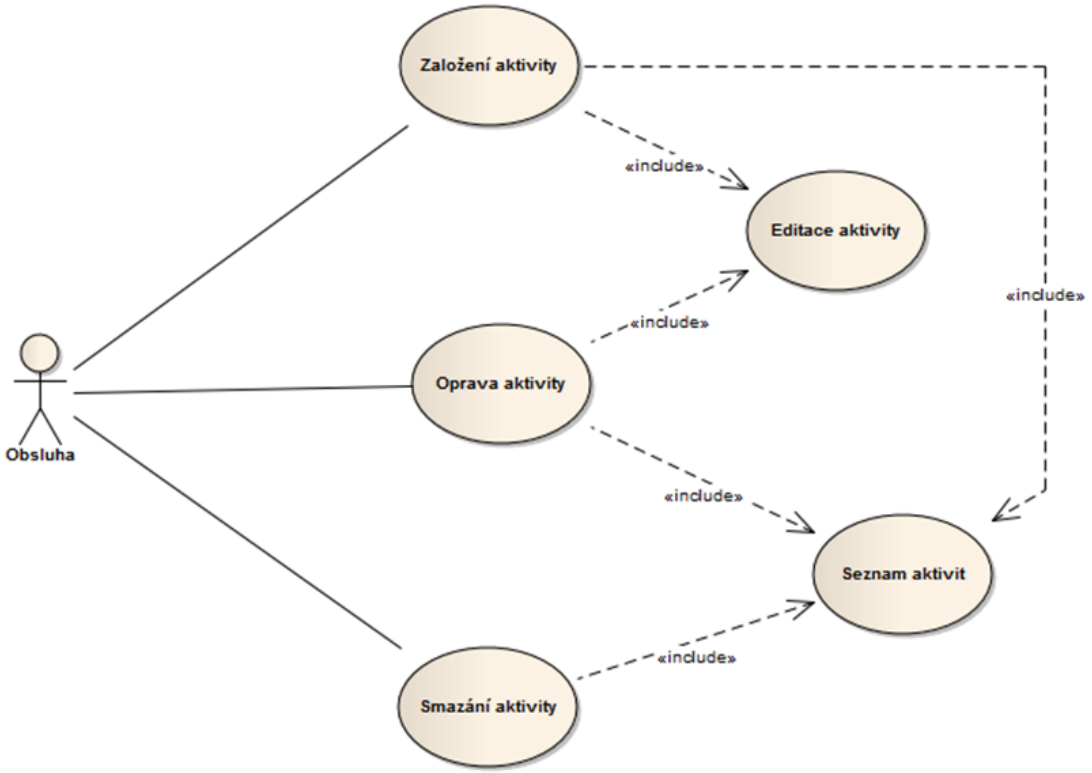
Obrázek 10: Náhled produktu – karta ceníky a číselníky

Vzhledem k tomu, že toto byl můj první zadaný úkol a na praxi jsem v tomto období docházel pouze tři dny v týdnu, časová náročnost byla výrazně větší než by se očekávalo. Vliv na to také měla má nezkušenost a tak se někdy stávalo, že jsem musel některé již naprogramované části naprogramovat znovu nebo jsem zjistil, že některé části kódu jsem vytvářel zbytečně.
V neposlední řadě hrál roli také rozsah zadání. Proto jsem tomuto úkolu věnoval přibližně pět týdnů.

6.2 Aktivity se zákazníkem nebo dodavatelem

Jako svůj druhý vývojový úkol jsem dostal úkol s názvem „Aktivity se zákazníkem“. Cílem bylo, aby obsluha mohla do systému zadávat nebo odebírat popis aktivit s daným zákazníkem a taktéž v nich účinně vyhledávat. Zobrazení těchto aktivit probíhalo na kartě zákazníka, ale také bylo možně je otevřít v samostatném okně. Později také přibyla možnost zobrazovat aktivity i pro dodavatele elektřiny.

Aktivity se zákazníkem či dodavatelem se z hlediska možností obsluhy nelišily od jiných zadání a proto můžeme vidět, že UML diagram je téměř identický jako diagram pro produkty. Jedinou změnou je změna popisku.



Obrázek 11: Základní UML diagram pro aktivity

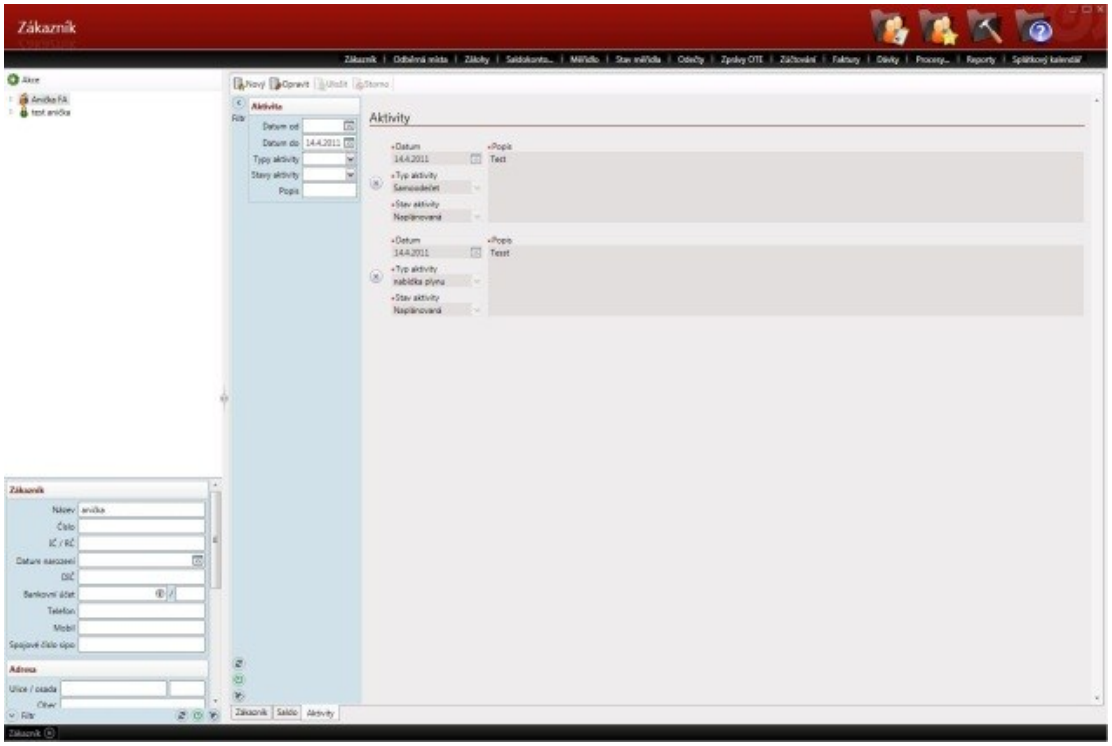
Tento úkol se lišil od většiny tím, že nezobrazoval právě jednu konkrétní položku, ale množství aktuálně zobrazených aktivit nebylo omezeno. Proto bylo v tomto případu vhodné použít StackPanel, na který byly vázány metody pro naplnění a získání dat.

StackPanel je komponenta, do které můžeme vložit řadu dalších komponent a StackPanel je seřadí vedle sebe nebo pod sebou. Pokud například odebereme nějaký prvek, který je umístěn mezi jinými, pak se následující přeskupí a zaplní jeho místo. To nám umožní dynamicky přidávat a ubírat prvky aniž bychom museli pokaždé nastavovat jejich pozici. StackPanel to udělá za nás.

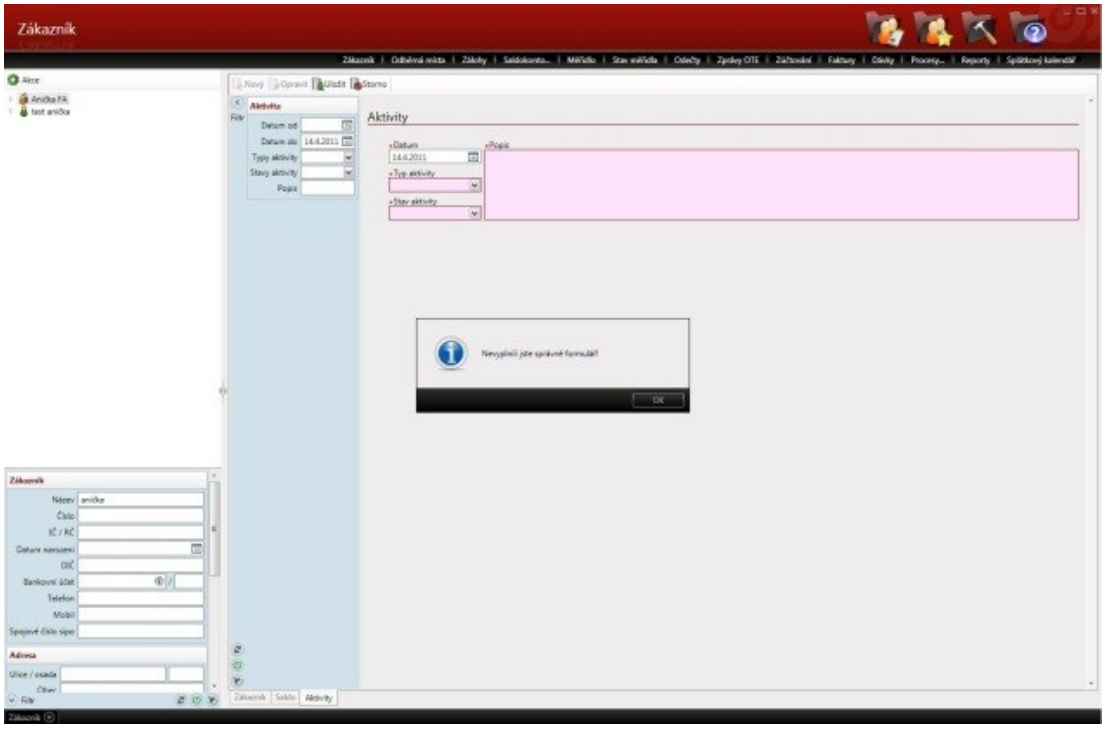
Po načtení byly aktivity předány metodě pro naplnění, která dynamicky vytvářela pro každou aktivitu ovládací prvek a jeden po druhém byly nastavovány jako potomek StackPanelu. Metoda pro získání dat sloužila k tomu, aby bylo možné upravené nebo nové aktivity z uživatelských prvků vytáhnout a poslat je na server k uložení.

Ovládací prvek jednotlivých aktivit v sobě také obsahoval automatickou metodu, která při získávání dat aktivity kontrolovala, zda je daná aktivita vyplněna správně. V tomto případě však pouze kontrolovala, zda není daná aktivity prázdná. Pokud by aktivita byla vyplněna nesprávně, daný prvek formuláře je zvýrazněn červeně a uživatel tak může snadno odhalit chybu.

Jak je z náhledů patrné, aktivity také obsahovaly přídavný filtr, aby se obsluha mohla v daných datech snadno orientovat za předpokladu, že aktivit bylo příliš mnoho.



Obrázek 13: Přehled aktivit



Obrázek 14: Špatně vyplněný formulář

Tento úkol byl přínosem v tom, jak načítat a zobrazovat data dynamicky. To s sebou ale nese další nevýhody a někdy i snadno přehlédnutelné chyby a proto mi byl tento úkol v rámci vývojového cyklu několikrát vrácen k opravě méně závažných chyb, které ale v programu nesměly zůstat pro finální nasazení.

Vyjádřit časovou náročnost není jednoduché, protože jsem v jeho průběhu onemocněl a nastoupil jsem opět až za tři měsíce, kdy jsem na něm dále pokračoval. Můžu ale říci, že oproti prvnímu úkolu jsem v tomto měl již mnohem lepší přehled a samotné programování také probíhalo rychleji.

7 Závěr

Tuto praxi cítím jako velikou programátorskou zkušenost. Než jsem nastoupil, těšil jsem se především na práci v týmu. Taky mě zajímalo, jak skutečné programování v praxi probíhá a čemu se vyhnout.

Když zpětně hodnotím kolektiv, říkám si, že o moc lépe už jsem dopadnout nemohl. Pracoval jsem v mladém, energickém a vitálním týmu. Kolegové nejen podněcovali dobrou mou náladu, ale také schopnost přemýšlet a samostatně pracovat. Když jsem například narazil na problém, snažili se mě navést na řešení tím, že mi poradili jak řešení hledat, nebo kde už někdo stejný problém řešil. Většinou tak plnili roli rádců, kteří pomůžou, když je třeba, ale nedělají mou práci za mě.

Díky tomuto přístupu a kvalitní organizaci práce a kódu - což také nemalou měrou přispívalo k tomu, aby práce byla bez konfliktů - můžu tuto odbornou praxi ze svého pohledu zhodnotit jako velmi vydařenou. Získal jsem mnoho zkušeností jak při psaní programů tak i při spolupráci s druhými.

Práce programátora mě zajímala již od dětství a i potom co jsem si vyzkoušel jaké to je být skutečný programátor-zaměstnanec, mě zájem o programování neopustil. I přes to, že bych se v životě nebránil jiným druhům zaměstnání, pozice vývojáře je pro mě stále lákavá.

I přes možnost dále s firmou D3Soft s.r.o. spolupracovat jsem se nakonec, kvůli časovým důvodům, rozhodl praxi ukončit. Vím ale, že až se budu ucházet o práci programátora, tato firma bude mezi prvními na mém seznamu.

Literatura

[1] D3Soft s.r.o.,
http://www.d3soft.cz/

[2] ISU
http://www.isleonardo.cz/produkty/leonardo-isu-billing.html

[3] Windows Communication Foundation
http://msdn.microsoft.com/en-us/netframework/aa663324

[4] Windows Presentation Foundation
http://msdn.microsoft.com/en-us/library/ms754130.aspx